Degree Project in Technology
Second cycle, 30 credits

A Side-Channel Attack on Masked and Shuffled Implementations of M-LWE and M-LWR Cryptography
A case study of Kyber and Saber

LINUS BACKLUND
A Side-Channel Attack on Masked and Shuffled Implementations of M-LWE and M-LWR Cryptography

A case study of Kyber and Saber

LINUS BACKLUND

Master’s Programme, Embedded Systems, 120 credits
Date: April 28, 2023

Supervisor: Kalle Ngo
Examiner: Elena Dubrova
   School of Electrical Engineering and Computer Science
Swedish title: En sidokanalsattack på implementationer av M-LWE- och M-LWR-kryptografi skyddade med maskering och slumpad operationsordning
Swedish subtitle: En studie av Kyber och Saber
Abstract

In response to the threat of a future, large-scale, quantum computer, the American National Institute of Standards and Technology (NIST) initiated a competition for designs of quantum-resistant cryptographic primitives. In 2022, the lattice-based Module-Learning With Errors (M-LWE) scheme Kyber emerged as the winner to be standardized. The standardization procedure and development of secure implementations call for thorough evaluation and research. One of the main threats to implementations of cryptographic algorithms today is Side-Channel Analysis (SCA), which is the topic of this thesis.

Previous work has presented successful power-based attacks on implementations of lattice cryptography protected by masking and even masking combined with shuffling. Shuffling makes SCA harder as the order of independent instructions is randomized, reducing the correlation between operations and power consumption. This randomization is commonly implemented by shuffling the order of the indexes used to iterate over a loop, using the modern Fisher-Yates algorithm. This work describes a new attack that defeats the shuffling countermeasure by first attacking the generation of the index permutation itself. The attack first recovers the positions of the first and last indexes, 0 and 255, and then rotates the encrypted messages using a ciphertext malleability applicable to many ring-based LWE schemes to shift two bits into the known positions from which they can be recovered. This procedure is repeated to recover full messages in 128 rotations.

The attack is tested and evaluated on masked and shuffled implementations of Kyber as well as Saber, another similar finalist of the NIST competition which is based on the Module-Learning With Rounding (M-LWR) problem. Compared to the previous attack on masked and shuffled Saber, which required 61,680 traces, the 4,608 needed for this attack demonstrates a 13-fold improvement.

Keywords

Public-key Cryptography, Post-Quantum Cryptography, Kyber, Saber, Side-Channel Attack, Power Analysis
ii | Abstract
Sammanfattning


Attacken har testats och utvärderats på implementationer, skyddade genom maskering kombinerad med slumpad operationsordning, av både Kyber och en liknande NIST-finalist, Saber. Jämfört med den tidigare attacken på Saber med samma skyddsåtgärder minskar den nya metoden det antal mätningar som krävs från 61,608 till 4,608, vilket motsvarar en 13-falding förbättring.

Nyckelord

Asymetrisk Kryptering, Kvantsäker Kryptografi, Kyber, Saber, Sidokanalsattack, Effektanalys
iv | Sammanfattning
Acknowledgments

I would like to thank Elena Dubrova for encouraging my deep interest in hardware security. Our long discussions on crazy ideas that may or may not work and your never ending enthusiasm about the topic of side-channels has kept me motivated. I would like to thank Kalle Ngo for providing support with whatever I needed, whenever I needed it. You have taught me a lot. I wish to thank Joel Gärtner for being the brilliant cryptologist that our team needed. Additionally, I want to thank all the rest of my colleagues at the hardware security group at KTH. It has been a joy and honor to work alongside such amazing people.

Stockholm, April 2023
Linus Backlund
# Contents

1 Introduction

1.1 Background .......................................................... 1
1.2 Problem ................................................................. 2
1.3 Purpose ................................................................. 2
1.4 Goals ................................................................. 3
1.5 Research methodology .............................................. 3
1.6 Delimitations ......................................................... 4
1.7 Structure of the thesis ............................................. 4

2 Background

2.1 Post-quantum cryptography ......................................... 5
  2.1.1 Kyber and Saber ................................................. 5
  2.1.2 Attack models ................................................. 6
  2.1.3 Ciphertext malleabilities .................................... 7
2.2 Side-channel analysis ............................................. 8
  2.2.1 Side-channels ................................................. 8
  2.2.2 Analysis ....................................................... 8
    2.2.2.1 Classical power analysis ............................ 9
    2.2.2.2 Welch’s t-test ........................................ 9
    2.2.2.3 Deep-learning ..................................... 10
  2.2.3 Countermeasures ........................................... 10
    2.2.3.1 Masking ............................................. 10
    2.2.3.2 Shuffling ........................................... 11
  2.2.4 Previous attacks ........................................... 11
2.3 Attack scenario .................................................. 12

3 Experimental setup

3.1 Target implementations .......................................... 15
3.2 Hardware setup .................................................. 16
# Contents

4 Profiling stage 19  
4.1 Trace acquisition ........................................... 19  
4.2 Trace pre-processing ...................................... 19  
4.3 Leakage analysis ........................................... 20  
4.4 Model training ............................................. 24  

5 Attack stage 29  
5.1 Attack design ................................................ 29  
5.2 Construction of CCTs ...................................... 30  
5.3 Cyclic rotation of CCTs ................................... 31  
5.4 Incremental ECCs .......................................... 32  

6 Results and Analysis 37  
6.1 Experimental results ...................................... 37  
6.2 Validity of results ......................................... 40  
6.3 Analysis ...................................................... 40  

7 Conclusions and Future work 43  
7.1 Conclusions ................................................ 43  
7.2 Future work ................................................ 44  

References 45
List of Figures

2.1 Pseudocode of CPA-PKE algorithms. [8] .......................... 6
2.2 Pseudocode of CCA-KEM algorithms. [8] .......................... 7
2.3 Non-authenticated key exchange. ................................. 13

3.1 Modified pseudo-C code of the masked_poly_tomsg() procedure in masked Kyber from [17] with shuffling added. [8] 16
3.2 Equipment for trace acquisition. [8] ................................. 17

4.1 An average trace representing the processing of masked message bits 64-72 in CRYSTALS-Kyber (top), t-test for share 1 (middle) and for share 2 (bottom). Both t-tests are performed on 1K traces. ................................. 21
4.2 An average trace representing the processing of masked message bits 64-72 in Saber (top), t-test for share 1 (middle) and for share 2 (bottom). Both t-tests are performed on 1K traces. ................................. 22
4.3 (a) An average trace representing the generation of the FY indexes 64-72 in CRYSTALS-Kyber (top) and its t-test (bottom); (b) An average trace representing the usage of the indexes during message processing (top) and its t-test (bottom). Both t-tests are performed on 5K traces. [8] .... 23
4.4 (a) An average trace representing the generation of the FY indexes 64-72 in Saber (top) and its t-test (bottom); (b) An average trace representing the usage of the indexes during message processing (top) and its t-test (bottom). Both t-tests are performed on 5K traces. [8] ..................... 24
4.5 Distribution of power consumption during the generation of FY indexes 64 and 128 (top) and 0 and 255 (bottom) at a trace point with the maximum absolute t-test value in Saber. [8] .......................... 25
4.6 Distribution of power consumption during the generation of FY indexes with the Hamming weight 0-8 at a trace point with the maximum absolute t-test value in Saber. [8] 26

6.1 Average undetected and detected errors depending on the number of repeated measurements $N$ for each code distance on Kyber. 39

6.2 Average undetected and detected errors depending on the number of repeated measurements $N$ for each code distance on Saber. 40
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>The MLP architecture. [8]</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>The MLP layer widths. [8]</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>CCT construction table for CRYS-TALS-Kyber. [8]</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>CCT construction table for Saber. [8]</td>
<td>35</td>
</tr>
<tr>
<td>6.1</td>
<td>Success rate of Kyber secret key recovery using an ECC with code distance $d$ and $N$ repetitions for 10 attacks. [8]</td>
<td>38</td>
</tr>
<tr>
<td>6.2</td>
<td>Success rate of Saber secret key recovery using an ECC with code distance $d$ and $N$ repetitions for 10 attacks. [8]</td>
<td>38</td>
</tr>
<tr>
<td>6.3</td>
<td>The number of traces required for successful key recovery in all 10 attacks. [8]</td>
<td>39</td>
</tr>
</tbody>
</table>
xii | List of Tables
## List of acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCA</td>
<td>Chosen Ciphertext Attack</td>
</tr>
<tr>
<td>CCT</td>
<td>Chosen Ciphertext</td>
</tr>
<tr>
<td>CPA</td>
<td>Correlation Power Analysis</td>
</tr>
<tr>
<td>DPA</td>
<td>Differential Power Analysis</td>
</tr>
<tr>
<td>ECC</td>
<td>Error-Correcting Code</td>
</tr>
<tr>
<td>EM</td>
<td>Electromagnetic Emanations</td>
</tr>
<tr>
<td>FO</td>
<td>Fujisaki-Okamoto</td>
</tr>
<tr>
<td>IND-CCA2</td>
<td>Indistinguishability under Adaptive Chosen Ciphertext Attack</td>
</tr>
<tr>
<td>IND-CPA</td>
<td>Indistinguishability under Chosen Plaintext Attack</td>
</tr>
<tr>
<td>KEM</td>
<td>Key Encapsulation Mechanism</td>
</tr>
<tr>
<td>LWE</td>
<td>Learning With Errors</td>
</tr>
<tr>
<td>LWR</td>
<td>Learning With Rounding</td>
</tr>
<tr>
<td>M-LWE</td>
<td>Module-Learning With Errors</td>
</tr>
<tr>
<td>M-LWR</td>
<td>Module-Learning With Rounding</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>PKE</td>
<td>Public-Key Encryption</td>
</tr>
<tr>
<td>PQ</td>
<td>Post-Quantum</td>
</tr>
<tr>
<td>PQC</td>
<td>Post-Quantum Cryptography</td>
</tr>
<tr>
<td>SCA</td>
<td>Side-Channel Analysis</td>
</tr>
<tr>
<td>SPA</td>
<td>Simple Power Analysis</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In this thesis, a new attack method for protected implementations of Module-Learning With Errors (M-LWE) and Module-Learning With Rounding (M-LWR) based Post-Quantum Cryptography (PQC) algorithms exploiting Side-Channel Analysis (SCA) on the Fisher-Yates shuffling algorithm is presented.

1.1 Background

As the development of large quantum computers progresses, it becomes increasingly urgent to develop and standardize new cryptographic algorithms which security assumptions will hold even in a Post-Quantum (PQ) era. The National Institute of Standards and Technology (NIST) introduced a competition in which several proposed schemes were evaluated. Among the four finalists were three lattice-based schemes, of which one (Kyber) was based on the M-LWE problem and another (Saber) on the very similar M-LWR problem. In July 2022, Kyber was chosen for standardization [1], meaning it has been selected as one of the cryptographic schemes of the future. Therefore, evaluating its implementations resistance to SCA is highly valuable.

One of the main threats to implementations of modern cryptographic algorithms is SCA. All computer algorithms, no matter how abstracted, eventually run on hardware. Hardware, as it is physical, inevitably leaks information about the operations or the data on which the operations are performed. A common side-channel is power. Power analysis exploits the fact that the power consumed by processors correlates with their operations and can be used to extract information for breaking the security of the algorithm.

There are countermeasures for SCA that try to hide the side-channel information and/or reduce its correlation with the secret values. One of these
is masking which separates the secret data into two or more shares that on their own hold no information, but can, after the processing, be combined into the final value. Masking is expensive to the implementation in terms of computation time and hardware resources. It has previously been shown that this countermeasure alone does not protect the algorithm against attacks [2, 3].

Another countermeasure is shuffling. Shuffling is a method of reducing the correlation between sensitive information and side-channel information by randomizing the order of independent operations. For example, the order in which elements of an array are processed. Previous attacks have shown that for M-LWR schemes, the shuffling countermeasure in combination with masking, is still possible to break, although the attack becomes significantly more difficult [4]. The attack method exploits a combination of SCA and a bit-flipping ciphertext malleability common in Learning With Errors (LWE) and Learning With Rounding (LWR) Public-Key Encryption (PKE) schemes such as those used in Kyber and Saber to recover the secret key.

1.2 Problem

As the previous attack has shown, the combined masking and shuffling countermeasures can be defeated through side-channels and malleability properties in the cryptographic schemes. The attack directly attacks the decryption operation through SCA and defeats the shuffling countermeasure using the ciphertext malleability to flip individual bits in the contained message.

It has however not been explored whether the shuffling permutation generation algorithm itself can be attacked with SCA. If it is possible to recover all or some of the indexes every execution, a similar attack to that of [4] could be done more efficiently.

The most common shuffling permutation algorithm is the popular and efficient Fisher-Yates shuffling algorithm. The problem is therefore to find whether the Fisher-Yates algorithm can be attacked through SCA to more efficiently break protected implementations of lattice-based PQC algorithms.

1.3 Purpose

As the standardization of Kyber is currently underway, now is the optimal time to evaluate its design and the security of current implementations. Finding weaknesses and addressing them should ideally be done as early as possible.
Prior to its potential mass adoption is when such discoveries have the most impact in terms of improving security and the least harm in terms of making vulnerabilities in deployed systems public. The net value of this project is expected to be good from an ethical standpoint.

The permutation generation of the shuffling countermeasure has, to the best of my knowledge, not previously been subject to power analysis. It is therefore of interest to anyone intending to use it as a protection mechanism in their implementations to know whether or not it can be attacked directly. As it is a simple mistake for developers to trust shuffling as a countermeasure without really understanding it, it is important to verify whether or not a shuffling countermeasure, not in itself protected, does in fact raise security.

1.4 Goals

The goal of the project is to analyze the Fisher-Yates shuffling algorithm from a side-channel security perspective and whether or not the results can be exploited in attacks against M-LWE and M-LWR schemes. This can be divided into the following sub-goals:


2. Demonstrate attacks on M-LWE and M-LWR schemes protected by masking and shuffling by first attacking the Fisher-Yates permutation generation.

1.5 Research methodology

The chosen approach for answering the research question can be divided into concrete steps.

1. Perform a literature study - What has already been done? How is SCA performed? How are the algorithms constructed? This step is important in order to perform the rest of the project in an efficient way with trustworthy results.

2. Implement shuffling in masked Kyber - As there is no masked and shuffled implementation of Kyber publicly available, shuffling must be integrated before the analysis can begin.
3. Perform power analysis on Fisher-Yates shuffling - Is the shuffling algorithm leaking exploitable information? This is an important phase of the project and a point in time where, should the answer be no, the research question can already be answered.

4. Construct attack method - Using the gained knowledge of the algorithms constructions and the results of the analysis of Fisher-Yates, construct a way to exploit the leaked information to break the implementation efficiently.

5. Execute attacks and collect results - Once the attack is constructed, perform several attacks on unseen datasets to collect unbiased results.

1.6 Delimitations

In order to keep the project manageable, this work will not:

- Analyze other shuffling algorithms than the Fisher-Yates algorithm.
- Analyze other side-channels than power consumption.
- Analyze attacks on other cryptographic schemes than Kyber and Saber.
- Try to find theoretical flaws in the analyzed algorithms.
- Analyze masking of higher order.
- Analyze any other countermeasures than masking and shuffling.
- Attack any proprietary products.

1.7 Structure of the thesis

Chapter 2 presents background on PQC, SCA and the work leading up to this thesis. Chapter 3 describes the experimental setup and what equipment was used in the experiments. The profiling stage, including analysis, processing, and training of networks is presented in Chapter 4, while the attack design and the techniques used to increase its efficiency are shown in Chapter 5. Finally, the thesis is concluded and future work is presented in Chapter 7.
Chapter 2

Background

This chapter provides some background on the post-quantum cryptography algorithms investigated in this work and explains some terminology and concepts of cryptology in general. Side-channels and ways of exploiting them, as well as relevant countermeasures are presented. Finally, previous work and the assumed attack scenario are described.

2.1 Post-quantum cryptography

The NIST introduced their Post-Quantum project, a competition for new cryptographic schemes that are meant to replace current public-key and key-exchange algorithms. The need for new algorithms is motivated by the advancements in the field of quantum computing and the fact that the hardness of the problems, which many of the algorithms used today rely upon, will be significantly reduced if or when a large and reliable enough quantum computer can be built. Using Shor’s algorithm [5], a quantum computer could perform integer factorization in polynomial time instead of sub-exponential time as is the case for conventional computers. The competition went on for three evaluation rounds before the first algorithm to be standardized, Kyber, was selected [1].

2.1.1 Kyber and Saber

Kyber and Saber, two of the third-round finalists, are both lattice-based cryptographic schemes. They are based on the M-LWE and M-LWR problems, respectively. The algorithms are very similar and primarily differ in their error sourcing, parameter choices, and polynomial representation.
Methods. They are designed to be used as Key Encapsulation Mechanisms (KEMs) but their specification covers their underlying PKE schemes as self-contained entities.

Figures 2.1 and 2.2 show a generalization of the algorithms for which Kyber and Saber can be viewed as different parametrizations. For more details, consult the specifications of Kyber [6] and Saber [7] respectively. There are three security levels defined for each of the schemes with corresponding parameter sets. In this work, we choose to analyze the “medium” levels, Kyber768 and Saber, supposed to provide a security comparable with that of AES-192. The presented analysis and attacks are however scalable to the other security levels as well. The parameter sets define the module rank \( k = 3 \) and the secret key coefficients in \( s \) to range from \(-2\) to \(2\) for Kyber and \(-4\) to \(4\) for Saber.

### 2.1.2 Attack models

The PKE schemes in Kyber and Saber themselves are only Indistinguishability under Chosen Plaintext Attack (IND-CPA) secure, while the KEMs are Indistinguishability under Adaptive Chosen Ciphertext Attack (IND-CCA2) secure. This is accomplished through the use of a modified Fujisaki-Okamoto (FO) transform which re-encrypts the decrypted message and compares the resulting ciphertext against the received one. If the decryption query does not pass the check, an output with no relation to the decrypted message is produced.
### CCA-KEM.KeyGen()

1: \( z \leftarrow \mathcal{U}(\{0, 1\}^{256}) \)
2: \((pk, s) = \text{CPA-PKE.KeyGen}())
3: \(sk = (s, pk, \mathcal{H}(pk), z)\)
4: \text{return} \((pk, sk)\)

### CCA-KEM.Encaps\((pk)\)

1: \( m \leftarrow \mathcal{U}(\{0, 1\}^{256}) \)
2: \((\hat{K}, r) = \mathcal{G}(m, \mathcal{H}(pk)) \)
3: \(c = \text{CPA-PKE.Enc}(pk, m, r)\)
4: \( K = \text{KDF}(\hat{K}, \mathcal{H}(c)) \)
5: \text{return} \((c, K)\)

### CCA-KEM.Decaps\((sk = (s, pk, \mathcal{H}(pk), z), c)\)

1: \( m' = \text{CPA-PKE.Dec}(s, c) \)
2: \((\hat{K}', r') = \mathcal{G}(m', \mathcal{H}(pk)) \)
3: \(c' = \text{CPA-PKE.Enc}(pk, m', r')\)
4: \text{if} \( c = c' \text{ then} \)
5: \text{return} \( K = \text{KDF}(\hat{K}, \mathcal{H}(c)) \)
6: \text{else} \)
7: \text{return} \( K = \text{KDF}(z, \mathcal{H}(c)) \)
8: \text{end if}

---

**Figure 2.2:** Pseudocode of CCA-KEM algorithms. [8]

### 2.1.3 Ciphertext malleabilities

A ciphertext malleability in cryptography is a property that allows a ciphertext to be manipulated in such a way that the contained message is modified in a predictable manner. The modified ciphertext will encrypt a message related to the original one but still unknown to the attacker. Previous work has shown that the PKE schemes of Kyber, Saber, and several others of the competition candidates have some of these malleability properties [9]. More specifically, the polynomials which the ciphertexts consist of can be manipulated in such ways that the bits in the encrypted message can be flipped, *i.e.*, a one becomes a zero and vice versa, or cyclically rotated, *i.e.*, the bits in the message shift a set number of steps wrapping around when they are shifted out. Ciphertext malleabilities are used in Chosen Ciphertext Attacks (CCAs) to control the behavior of the decryption algorithms and derive information through, for
example, observing decryption failures or physical side-channels.

### 2.2 Side-channel analysis

**SCA** can be used as a non-invasive attack and is one of the main threats to cryptographic implementations today. Even though a cryptographic scheme is theoretically secure, its implementation must run on a physical system, making it susceptible to side-channel-based attacks.

Side-channels allow for not only the final values to be stolen but also intermediate ones as well. It is because of this that the **IND-CCA2** security of schemes such as Kyber and Saber can be bypassed, making **CCAs** possible.

#### 2.2.1 Side-channels

Side-channels are unintended sources of information. The information found in side-channels is often considered “leaking”, giving the name to the term “side-channel leakage”. On an algorithmic level, decryption failures influenced by some secret value can be considered a side-channel. In complex computer systems, software side-channels such as timing differences caused by branching in code, cache usage, or even relatively low-level hardware design such as branch prediction can be found. For embedded systems, architectural features such as caches are not as common. A sound implementation with constant time execution is therefore not likely to be vulnerable to timing attacks. However, there are still plenty of physical side-channels that exist even in embedded systems. Power analysis relies on the fact that the power consumption of processors depends on the operations and the data those operations are carried out on [10, 11]. The currents within a chip cause **Electromagnetic Emanations (EM)** that can be analyzed to derive information on the operations or data. If analog radio transmission circuitry and some digital processor share a power supply, the power-based side-channel can even leak through the transmission and be exploited at a distance [12]. There are other physical side-channels such as acoustics from electrical noise or heat of some components although they are not as often utilized in attacks.

#### 2.2.2 Analysis

Side-channels leak information about the operations of an implementation. They can be analyzed to derive information that is directly or indirectly related to the information of interest.
2.2.2.1 Classical power analysis

The focus of this work is power analysis. This section, therefore, summarizes a few of the classical categories of power analysis.

**Simple Power Analysis (SPA)** is, as its name suggests, very straightforward. By visually examining a power trace, data can be derived by metrics such as the width or height of features. For example, in vulnerable implementations of the RSA algorithm, for each bit in the key, a square and multiply are performed if the bit is one, and only square if the bit is zero. This leads to visible differences between the two cases and the possibility of a timing attack [10].

**Differential Power Analysis (DPA)** is a statistical method of deriving information through power consumption. While variations in executed instructions are generally very visible on power traces, variations caused by the data are very small. For constant-time implementations protected against SPA, DPA can be used [11]. By collecting a larger set of traces, if the implementation is fully deterministic, noise can be eliminated by averaging the data points. A sufficient reduction of noise will reveal any small power consumption differences caused by the secret data.

**Correlation Power Analysis (CPA)** is a more sophisticated analysis method, building on DPA, for which knowledge about the algorithm under attack is required. By first defining a power hypothesis, meaning a hypothesis of how the power consumption depends on the secret as well as guess of what the secret might be, it can be correlated with the acquired data. This process is repeated for each possible value of the secret. The hypothesis which correlates the strongest with the data is most probable to be the correct one. The S-box step of the first and last round of AES can be attacked with this method to recover one secret key byte at a time [13].

2.2.2.2 Welch’s t-test

Side-channel leakage in unprotected implementations can be found by correlating collected measurements with known labels against a leakage model. For power analysis, measurements of the power consumption of a chip are collected during the execution of the implementation. The power traces can then be analyzed with Welch’s t-test [14]. The collection of traces is split into two sets, \( T_0 \) and \( T_1 \), according to a power model believed to correlate with the power consumed. Using Equation (2.1) a t-test is calculated. The
t-test indicates at which sample points in the traces a statistical difference in means can be observed. A common power model for microprocessors is the Hamming weight (number of ones) of the data being processed. This leakage identification method requires the traces to be perfectly synchronized and that the values in the execution correlate with the values used as labels.

\[
t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{\sigma_0^2}{|T_0|} + \frac{\sigma_1^2}{|T_1|}}},
\]

(2.1)

### 2.2.2.3 Deep-learning

A recent method for recovering secrets via side-channels is through the use of deep learning. Training deep Neural Networks (NNs) on data such as power traces has proved very efficient compared to classical methods. It significantly lowered the bar for how deep analysis and understanding of the target implementation was required to perform an attack. While typically requiring a large number of traces for training, not necessarily from the target device, the number of traces required for the actual attack can be reduced by as much as two orders of magnitude \[15\]. Combining deep learning with properties such as ciphertext malleabilities has been proven to be able to transform non-differential attacks into differential ones \[16\].

### 2.2.3 Countermeasures

The threat of side-channel attacks has motivated the development of countermeasures. Ideally, a countermeasure would entirely remove the side-channel leakage causing the vulnerabilities in an implementation. This can sometimes be done for some channels such as timing. By changing the implementation to execute in constant time, the vulnerability can be entirely eliminated. However, for other side-channels such as power consumption, the leaked information can not be completely eliminated without completely changing the hardware. Instead, it is common to try to hide the leakage or reduce its correlation with sensitive information.

#### 2.2.3.1 Masking

The masking countermeasure is a means of dividing data into multiple shares that can be processed individually while still, when combined, represent the final, processed data. The idea behind this countermeasure is that no individual share, at any point during the operations, will reveal any information about the
masked secret. Boolean and arithmetic are two common types of masking. Which one is used depends on the operations to be carried out. Sometimes it becomes necessary to convert the masked data from one domain into another if the operations can not be performed in the current one. Masking, while effective in hiding side-channel leakage from methods such as the t-test, comes at the cost of significant resource overhead. The first-order masked versions of Kyber and Saber used in this work have an execution time overhead of 250% and 150% over their unmasked counterparts, respectively [17, 18]. Previous work has shown that through deep learning, the secret shares can be extracted individually and then combined [9] or even extracted and combined simultaneously by a single NN [2].

2.2.3.2 Shuffling

The shuffling countermeasure is a technique for reducing the correlation between the secret and time. By randomizing the order in which independent operations are carried out, the complexity of a potential attack can be increased. The cost of shuffling is relatively small. If the algorithm already is structured to iteratively operate on indexed data, the order of indexing can simply be exchanged for a randomized permutation. Such a permutation is commonly generated by modern Fisher-Yates [19], a simple and very efficient shuffling algorithm. In an attack by Ngo, et al., [4] an earlier version of the masked and shuffled implementation of Saber considered in this work was successfully attacked by exploiting the bit-flipping ciphertext malleability and recovering the Hamming weight of the entire decrypted message. By first recovering the Hamming weight of the original message, the algorithm could be prompted with manipulated ciphertexts in which the bits were flipped, one at a time, and the Hamming weight recovered from those modified messages could be compared with the original one to see whether it increased or decreased.

2.2.4 Previous attacks

Implementations of both Kyber and Saber have been successfully attacked through SCA in the past to recover both messages and, through the use of Chosen Ciphertexts (CCTs), long-term secret keys. The attack of [9] showed that implementations protected by first-order masking can be compromised by recovering each share individually. Later, the attack of [2] showed a practical attack on the first-order masked Saber implementation of [18] using deep learning where the NNs were able to recover both shares simultaneously.
and XOR them together. The work of [2] also introduced the construction of CCTs based on Error-Correcting Codes (ECCs) in order to recover the secret key more efficiently. The attack was extended to the higher-order masked implementation of [20] in [21] and later [22] presented a successful attack. In [3], the same attack was shown on the masked implementation of Kyber from [17] using the previously known vulnerability found in [23]. The attack on a first-order masked and shuffled implementation of Saber presented in [4] is the closest related attack to the ones presented in this work. It utilizes the same technique for directly recovering the unmasked values used in [2] to defeat the masking countermeasure as well as the bit-flipping ciphertext malleability found in [9] to defeat the shuffling countermeasure. By recovering the Hamming weight of the encapsulated message, which is independent of the shuffling order, and that of modified messages where one bit is flipped at a time, the Hamming weights can be compared to derive the message bit. This attack was shown successful but required a significantly higher number of traces than attacks on unprotected and masked-only implementations.

The only previous work that has made use of the cyclic rotation ciphertext malleability is [24] where session keys of an unprotected implementation of Kyber were recovered through a template attack and rotated ciphertexts.

### 2.3 Attack scenario

The attack scenario assumes the simplest key exchange scenario depicted in Figure 2.3 where the device under attack, the client, establishes a shared session key with a server without authentication. The client holds the keypair and performs decapsulations using the secret key. The attack scenario assumes that the attacker has physical access to the device under attack and has the ability to query the algorithm with ciphertexts for decapsulation. Additionally, it assumes another, identical device, over which the attacker has full control and can use for profiling. The goal of the attacker is to extract the secret key of the client device.
(pk, sk) = KEM.KeyGen()

key = KEM.Decaps(sk, c)

(c, key) = KEM.Encaps(pk)

Figure 2.3: Non-authenticated key exchange.
Chapter 3

Experimental setup

This chapter describes the implementations used in the experiments as well as the hardware used to collect the data.

3.1 Target implementations

The masked and shuffled implementation of Saber used in this work is the one from [4], updated to the latest version. It is a shuffled adaptation of the masked implementation of [18] which is optimized for the ARM Cortex-M4 microprocessors.

To the best of our knowledge, no masked and shuffled implementation of Kyber exists. Therefore, the implementation of shuffling presented in [4] is adapted to the masked-only, Cortex-M4 optimized, Kyber implementation of [17]. At the end of the decryption, the function masked_poly_tomsg, decodes a message from the polynomial domain into a bit-representation. The function is masked and contains a representation conversion from arithmetic to boolean masking. The operations on message bits are performed in a loop and are independent of those on other bits. By simply changing the array indexes used during the loop to a randomly generated permutation instead of the loop iterator, the processing order can be randomized. The Fisher-Yates shuffling algorithm generates the random permutation, just as in [4] for Saber. The pseudo-C code of the shuffling implementation in the masked_poly_tomsg function, decoding from polynomials to bits, which contains the message vulnerability of interest, is presented in Figure 3.1. The blue lines highlight the principal vulnerable lines exploited for the index recovery and the red lines highlight the main instructions leaking the message bits.

The firmware for the implementations was compiled with the highest
### 3.2 Hardware setup

The hardware used for power trace acquisition is the *ChipWhisperer-Pro (CW1200)* capturing device and the *STM32F4* target board connected through the *CW308 UFO* adapter board. The target is based on the ARM Cortex-M4 core for which the masked and shuffled software implementations considered in this work are optimized. Two target boards are used, $D_P$ and a significantly more worn out $D_A$. To further illustrate the attacks resilience to manufacturing
diversity, the boards were acquired from different chip vendors. $D_P$ is used for profiling while $D_A$ is used during the attacks. As seen in Figure 3.2, the target board is connected to the capturing device through the adapter board; the capturing device is in turn connected to a laptop. The capturing device is capable of recording up to 98k samples at 105MS/s. The target devices are run at 24MHz.
18 | Experimental setup
Chapter 4

Profiling stage

The following chapter describes the process for trace acquisition, pre-processing and analysis. It is concluded with a description of how the NN models are trained and used.

4.1 Trace acquisition

The newer of the two target boards, $D_P$ is used as the profiling device. Power traces for Kyber are collected during the shuffling permutation generation as well as the masked_poly_tomsg decoding procedure which additionally contains a conversion from arithmetic to boolean masking. The Saber traces contain the index permutation generation and the poly_A2A procedure which performs masked logical shifting on arithmetic shares. In other words, the collected traces represent the execution of the function that generates the shuffling indexes as well as the function that leaks the message bits, in which the indexes are used. The Saber traces are captured at 3S/clock cycle, where the trace buffer of the ChipWhisperer capturing device is the limiting factor. The operations recorded in Kyber are significantly longer. The constraint of fitting the traces within the sample buffer limits the capture rate to 1S/clock cycle.

4.2 Trace pre-processing

The compiled shuffling permutation generation did not execute in perfectly constant time. The reason for this non-deterministic behavior is the in-loop sourcing of randomness. The built-in random number generator of the
STM32F4 may require up to 40 cycles to generate a new random number. The shuffling generation loop is faster than 40 cycles, leading to a sporadic and random delay in each iteration. The shuffling traces were synchronized using correlation analysis. As can be seen in Figures 4.3 and 4.4, the traces contain repeating patterns. There are 255 repetitions, each representing the selection of an index. The 256th index is implied when all others are set as it is the last remaining one. Due to the three-stage pipeline of the Cortex-M4, the first and last repetitions look somewhat different from the rest. Therefore, the mean of the second repetition for all traces is used as the pattern for correlation. At each peak in the cross-correlation between each trace and the pattern, a fixed-length segment is cut out from the trace and appended to the new, synchronized one.

### 4.3 Leakage analysis

To find and analyze the leakage, Welch’s t-test Section 2.2.2.2 is used. Subsets of the power traces in $T$ are created by selecting traces according to their decrypted message bits for one masked share $a$ at a time as in Equation (4.1). The $j$th trace is appended to either $T_0$ or $T_1$ depending on whether the $i$th processed bit of the decrypted message share $m_a$ is 0 or 1. The $i$th index of the $j$th shuffling permutation represents the index of the $i$th bit processed. This can be seen as performing a typical t-test on one shuffled message share at a time in order to visualize the message leakage despite the shuffling and boolean masking countermeasures. This is of course not possible in a real attack scenario but is not necessary to perform the attack due to the direct unmasking and recovery of message bits shown in [2].

$$T_0 = \{ T_j \in T \mid m_a[FY_j[i]] == 0 \},$$
$$T_1 = \{ T_j \in T \mid m_a[FY_j[i]] == 1 \}, \quad (4.1)$$

Similarly, the t-tests for the shuffling indexes themselves are produced using the criteria in Equation (4.2) where the $j$th trace is appended to $T_0$, $T_1$ or neither depending on whether the Hamming weight of the $i$th index in the $j$th permutation is less than or greater than 4. This is done for both the traces of the permutation generation and those of the procedure where the indexes are used.

$$T_0 = \{ T_j \in T \mid HW(FY_j[i]) < 4 \},$$
$$T_1 = \{ T_j \in T \mid HW(FY_j[i]) > 4 \}, \quad (4.2)$$

Figures 4.1 and 4.2 show the averaged traces as well as the t-tests,
Figure 4.1: An average trace representing the processing of masked message bits 64-72 in CRYSTALS-Kyber (top), t-test for share 1 (middle) and for share 2 (bottom). Both t-tests are performed on 1K traces.

...performed on 1k traces, of bits 64-72 of the message shares for Kyber and Saber respectively. It can be noted that the leakage of the shares does not overlap completely. This is expected as they are processed sequentially. It can further be observed that the leakage in Kyber is significantly weaker than that in Saber. This is caused by the nature of the vulnerabilities. The vulnerability in Saber leaks bit-wise, *i.e.*, a single message bit at a time, while the leakage in Kyber is accumulative, *i.e.*, the message *byte* is read from memory, updated with the processed bit, and written back again. For a deeper
analysis of accumulative leakage, the reader is referred to the first paper on defeating masked Saber [2]. Simply put, accumulative leakage means that all the previously set bits of a memory segment (in this case a byte) leak simultaneously as the currently set bit. The current bit, therefore, leaks less as it is hard to distinguish its leakage from the other ones. If the previously set bits are known, this can be compensated for. When shuffling is used, however, which bits have previously been set in the message byte is random. This leads to the t-test showing an average of the accumulative leakage.
In Figures 4.3 and 4.4, the averaged traces and the respective t-tests, performed on 5k traces, for the generation and usage of indexes 64-72 are shown. The generation of the indexes leaks significantly stronger than their in-loop usage. The leakage during generation is greater in Saber than in Kyber. The assembly of the two compiled Fisher-Yates implementations were compared and the only difference was address offsets. The cause for the greater leakage is therefore assumed to be the higher sampling rate.

For bit-recovery, the NNs simply has to distinguish between two values. For complete byte recovery, which is the case for the shuffling indexes, distinguishing between 256 different values is necessary. To analyze the possibility of distinguishing between indexes, the distributions of sample values at the point of strongest leakage during the permutation generation in Saber are plotted. In Figure 4.5, the distributions for indexes 64 and 128, both with a Hamming weight of 1, as well as indexes 0 and 255, with Hamming weights of 0 and 8 respectively, are depicted. The distributions of indexes with the same Hamming weight are almost completely overlapping which means that distinguishing between them is almost impossible. For those of very different Hamming weights, the observation is the opposite. When performing
the t-test, the power model that is used assumes that the Hamming weight of the value correlates with the power consumption. The results of Figure 4.5 comply with this assumption and suggests that distinguishing between values with the same Hamming weight is hard. In Figure 4.6, the distributions of sample values for indexes with the same Hamming weights show that distinguishing between groups of values, according to their Hamming weight, is significantly simpler. This analysis shows that the only two values that can be reliably recovered are index 0 and 255 which are the only values of their Hamming weights, 0 and 8 respectively. In extension, this means that the position of the first and last message bits, in the shuffled processing, can be reliably identified by exploiting side-channel leakage from the Fisher-Yates algorithm.

### 4.4 Model training

Index prediction, even when only distinguishing the Hamming weight, is harder than predicting message bits. In order to provide the NN models with as much information as possible, the trace segment where an index is
generated, as well as the one where it is used, is provided as input to the models for training and classification. Additionally, the segments representing the indexes set before and after, as well as the indexes used before and after, i.e., the neighboring segments, are provided. The reasoning behind this is that values that are already in registers can leak again and affect the leakage of the new values when being overwritten. The inclusion of these neighboring segments provides some contextual information about the state to the models and slightly improved the classification accuracy.
Figure 4.6: Distribution of power consumption during the generation of FY indexes with the Hamming weight 0-8 at a trace point with the maximum absolute t-test value in Saber. [8]

For the Saber traces, a method of trimming redundant sample points helped improve accuracy further. By first training five NNs each for indexes and message bits and then using the stuck-at-0 fault method of [25], samples that have no apparent significance to the networks can be identified. The networks were first evaluated on a testing set, then evaluated again, with one sample zeroed at a time. If the prediction accuracy dropped less than 0.5% for indexes or 0.01% for messages, they were not used in the final training or classification. The trimming was tested on Kyber as well but no improvement in prediction accuracy could be observed. The greater improvement for Saber is believed to be due to the higher redundancy caused by three times oversampling. By removing some of that redundancy, the models could learn more effectively what information is valuable.

The NN model architecture used is a standard Multilayer Perceptron (MLP). Table 4.1 shows the generic architecture and Table 4.2 lists the layer widths for the specific models. Other layer widths were explored but the ones listed performed the best.

Ten models for indexes and ten models for message bits were trained on 50k traces for Kyber and 15k traces for Saber. The cut-and-join method of [4] was used to increase the training set by a factor of 253 by training on each
of the indexes and message bits except the first and last ones. Due to the three-stage pipeline of the processor, the first and last processed indexes and message bits in the traces look different than the others. Therefore ten models are trained for the first processed index and message bits respectively. The same is done for the ones processed last. For Kyber, where the message bit leakage is accumulative, ten models were trained per bit position within a byte. The bit position is calculated as $i \mod 8$ where $i$ is the index of bit $m[i]$ in the message $m$. Since the bits are stored in a packed byte array, the leakage varies slightly depending on the bit position. By training models for each bit position, accuracy could be slightly increased.

The predictions of the model ensembles are combined differently for index prediction and for message bit prediction. For index prediction, the probability vectors of the models are simply multiplied. For message bit prediction, the predictions were combined through majority voting, excluding any model yielding confidence less than 90%. This proved to somewhat improve the reliability of the message bit predictions.
Profiling stage
Chapter 5

Attack stage

This chapter describes the final attack design. It gives the general methodology, details on how the CCTs were constructed, how they were rotated and finally how they were designed to make up incremental ECCs.

5.1 Attack design

In the ideal scenario for an attacker, all indexes of the shuffling permutation would be possible to recover. This would allow a complete bypass of the shuffling countermeasure and all message bits could be extracted. However, since only two indexes, 0 and 255, can be recovered, only the first and last bit of the message can be retrieved. In the previous attack on a masked and shuffled implementation of Saber [4], a method using the bit-flipping ciphertext malleability was used to extract a single bit at a time. Using the other ciphertext malleability discovered in [9] for message rotation, applicable to ring-based LWE and LWR, any two adjacent message bits could be rotated into the recoverable positions. After 128 rotations, the full message, and in extension the session key, is extracted. Furthermore, the recovery of a bit in the previous attack required the correct recovery of all message bits as it was derived from the message Hamming weight. By first attacking the permutation generation, the recovery of a bit requires only the correct recovery of the index and the single bit itself.

To recover the secret key, cyclically rotated CCTs had to be created. Due to asymmetries in the algorithm design, the cyclic rotation malleability was not directly applicable to the specifically created ciphertexts. Instead, the theory of rotating the message was incorporated into the construction method itself which is explained in Section 5.2.
As in the previous attack, this work makes use of ECCs. In addition to using codes of greater code distance, providing greater detection and correction capabilities, this work presents a new method of using incremental ECCs allowing an attack to be extended to use more powerful codes. This is further explained in Section 5.3.

In total, an attack requires $3 \cdot l \cdot 128 \cdot N$ traces, where the factor 3 represents the three secret key polynomials defined by the security level and therefore the parametrization of the algorithms, $l$ is the codeword length, the same as the number of ciphertexts required for the ECC used, 128 is the number of rotations needed of each ciphertext, and $N$ is repetitions of measurements of each ciphertext.

### 5.2 Construction of CCTs

The CCTs, before they are rotated, are constructed very similarly to the ones used in previous works. The secret key, in the security levels of Kyber and Saber studied in this work, consists of three polynomials $r \in \{0, 1, 2\}$. The ciphertexts, for both schemes, consists of $u$ and $v$ where $u$ is three polynomials and $v$ is a single polynomial. The chosen ciphertext polynomials can be described by Equation (5.1) and Equation (5.2) where the constants $k_0$ and $k_1$ are specifically chosen to produce chosen patterns in the decrypted message bits.

$$u = \begin{cases} 
(k_1, 0, 0) \in \mathbb{R}^{3\times1}_q, & \text{for } r = 0 \\
(0, k_1, 0) \in \mathbb{R}^{3\times1}_q, & \text{for } r = 1 \\
(0, 0, k_1) \in \mathbb{R}^{3\times1}_q, & \text{for } r = 2 
\end{cases} \quad (5.1)$$

$$v = \begin{cases} 
k_0 + (k_2 \sum_{i=1}^{254} X^i) + k_0 X^{255}, & \text{for Kyber} \\
k_0 \sum_{i=0}^{255} X^i, & \text{for Saber} 
\end{cases} \quad (5.2)$$

For Saber, the construction is the same as that used in [26] and [9] but with new constants. For Kyber however, the new constant $k_2$ is introduced to minimize the probability that the decrypted bits that are not being recovered, i.e., the bits that are not in position 0 or 255, decrypt to ones. This essentially minimizes the Hamming weight of the message. Reducing the number of ones in the message helps reduce the interference from previously decrypted bits, caused by the accumulative leakage, on the bit being recovered.

The decryption can be somewhat abstracted and simplified from Section 2.1.1 to $m = v - s \cdot u$. By constructing the sparse ciphertexts as
in Equations (5.1) and (5.2), the $i$th bit in the decrypted message $m[i]$ depends exclusively on the constants $k_0$, $k_1$, $k_2$, and the $i$th secret coefficient $s_r[i]$ in the targeted secret key polynomial. For the ciphertexts described, this can be shown by writing the expression for each of the message bits as in Equations (5.3) and (5.4), where all but $k_0$, $k_1$, $k_2$, and $s_r[i]$ are constants of the respective schemes.

**Kyber:** $m[i] = \begin{cases} \frac{2}{q} \cdot \left( \left\lfloor \frac{k_0}{2q} \right\rfloor - s_r[i] \cdot \left\lfloor \frac{k_1}{2q} \right\rfloor \right) \mod q \right) \mod 2, & \text{if } i \in \{0, 255\} \\
\frac{2}{q} \cdot \left( \left\lfloor \frac{k_0}{2q} \right\rfloor - s_r[i] \cdot \left\lfloor \frac{k_1}{2q} \right\rfloor \right) \mod q \right) \mod 2, & \text{otherwise} 
\end{cases}$ (5.3)

**Saber:** $m[i] = \left( \left( k_1 \cdot (s_r[i] \mod p) - 2^{r_p-c_T} \cdot k_0 \right) \mod p \right) \gg (\epsilon_p - 1)$ (5.4)

### 5.3 Cyclic rotation of CCTs

Cyclically rotating encrypted messages by manipulation of the ciphertext for ring-based LWE and LWR schemes was presented in [9]. While the method works on properly generated ciphertexts, the chosen ciphertexts do not always rotate correctly. The decode function of the decryption algorithm is not completely symmetric, for neither Kyber nor Saber, sometimes causing the bits that wrap around to incorrectly flip. This is caused by decode(x) and decode(−x) not always decoding the same. The rotations of the CCTs are therefore adapted to the way they are constructed.

In the original message rotation method, by multiplying the ciphertext polynomials of $u$ and $v$ with the indeterminant $X$, the polynomial decoding to the decrypted message is rotated, cyclically rotating the message bits. The rings of the ciphertext polynomials are negacyclic, meaning that when their coefficients fall over the "edge", they wrap around to the beginning again with their sign flipped. For example, if the polynomial $c0 + c1 \cdot X + c2 \cdot X^2 + \ldots + c254 \cdot X^{254} + c255 \cdot X^{255}$ is multiplied with $X$, the resulting polynomial is $-c255 + c0 \cdot X + c1 \cdot X^2 + \ldots + c253 \cdot X^{254} + c254 \cdot X^{255}$. In the chosen ciphertexts, these negated coefficients are undesired as they are no longer the carefully chosen values of $k_0$, $k_1$, and $k_2$ in Equations (5.3) and (5.4) if their signs flip. This causes, due to the asymmetries in decode, undesired bit-flips. Furthermore, for Kyber, the positions of $k_0$ and $k_2$ in $v$, as shown in Equation (5.2), are adapted to the two indexes to be recovered, 0 and 255,
and should therefore not be changed. The goal is to rotate only the single secret key coefficients depended upon in the equations while preserving \(k_0, k_1,\) and \(k_2\) so that the dependency of message bit \(m[i]\) changes from \(s_r[i]\) to \(s_r[i + 1]\).

To accomplish this, only \(u\) is rotated. To rotate further than one bit, the process can be repeated or, equivalently, the polynomials can be multiplied with \(X^n\), where \(n < 256\) is the number of bits rotation. The number of rotated bits should be less than 256, i.e., less than a full rotation, to ensure \(k_1\) in \(u\) does not flip sign.

Using ciphertexts rotated one bit position in this way, the bit \(m_0[0]\) will, instead of depending on \(s_r[0]\), depend upon \(-s_r[255]\), the negated, wrapped around \(s_r[255]\). This is due to the negacyclic polynomials and must be taken into account when deriving the secret coefficients from the decrypted messages.

In the experiments, it was revealed that there are slight differences in the decryption between the masked implementation of Kyber and its unmasked reference implementation. The constants \(k_0, k_1,\) and \(k_2\) were therefore chosen such that \(\text{decode}(x \pm \varepsilon) = \text{decode}(x)\) for a small \(\varepsilon\), a sufficient noise margin guarantees identical decryption in both implementations.

### 5.4 Incremental ECCs

By the use of CCTs, the concept of “divide and conquer” can be applied to recover a single secret key coefficient at a time, per message bit. This is done by making each decrypted message bit depend on only one secret key coefficient. Thus one key coefficient can be recovered at a time, per recoverable message bit. The work of [2] introduced the method of selecting a set of CCTs such that their decrypted message bits form unique codewords, depending on the secret coefficient, that also is a codeword of an ECC. An \([8,4,4]\) extended Hamming code was used. In the notation \([l,w,d]\) it has a codeword length of 8, a dataword length of 4, and a code distance, i.e., minimum Hamming weight between any two codewords, of 4. As such, at the cost of some redundancy, it can correct one erroneous bit and detect one additional error bit. This means that the secret coefficient can be recovered even if one of the decrypted message bits of the codeword is incorrectly recovered. Furthermore, if an error is not possible to correct but is detected, the attacker knows the recovered coefficient is still unknown. Detecting errors essentially reduces “false positives”; allowing for brute-forcing or enabling cryptological attacks on the remaining unrecovered coefficients. Not detecting an error makes theoretical attacks harder as there is no way for the attacker to
know which coefficients are incorrect. The error-correcting approach proved significantly more efficient than simply collecting more traces of the same message, greatly reducing the number of traces needed.

In this work, the CCTs used in the attack are also based on error-correcting linear codes. In contrast to the work of [2], the impact of the code distance, i.e., the error correction capabilities of the code, is explored. Further, the new concept of incremental codes, where each set of CCTs forming the code of code distance $d$ is a subset of the set that makes up the code with distance $d + 1$. The incremental codes allow an attacker to capture the relatively few traces needed for an attack of low code distance first. If access time to the target device allows, the captured trace set can be extended to construct a code of larger code distance without having to start over. The experiments of this work showed that there is an optimal code distance, which is not known in advanced. Capturing traces for an incremental code of greater distance allows an attacker to try recovery using codes of less distance as well. Tables 5.1 and 5.2 lists the CCT constants, their decrypted bits for each possible secret coefficient, and their mapping within the ECCs.

The first multi-column of Tables 5.1 and 5.2 gives the indexes defining the order of the CCTs used in the codewords of the code with distance $d$. The second column defines the constants used in constructing the CCTs. The third multi-column shows the patterns generated by the decrypted message bits for each possible value of the secret key coefficient they depend on. For the parametrizations used, there are 5 possible values for the secret coefficients in Kyber and 9 in Saber. This means that the necessary dataword length $w$ is $\lceil \log_2 5 \rceil = 3$ for Kyber and $\lceil \log_2 9 \rceil = 4$ for Saber. The codeword length $l$ as the minimum bounds (the smallest possible) and is represented by the number of CCT indexes for each $d$.

As an example, to recover the first coefficient of secret key polynomial $r = 0$ in Kyber using an ECC with distance $d = 2$, having the capability of detecting one error and correcting none, ciphertexts are constructed as in Equations (5.1) and (5.2) choosing the constants $(k_2,k_1,k_0)$ according to the indexes under the appropriate column in Table 5.1. Re-ordering the ciphertexts using the indexes yields the constant-tuples $(2,335,15), (0,77,5), (3,432,8),$ and $(3,606,3)$. If the first bits of their corresponding decrypted messages are $(0,1,1,0)$, by referencing the message bit-to-coefficient mapping in the table, the first secret key coefficient is recovered as $-2$. If instead, the bits are $(1,1,1,0)$, which does not match any mapping, an error has been detected.

Because the dataword length is able to represent more than the possible secret coefficients, not all datawords are used. Therefore, if an unused
Table 5.1: CCT construction table for CRYSTALS-Kyber. [8]

<table>
<thead>
<tr>
<th>Order of codeword bits for a code with distance d</th>
<th>CCT constants</th>
<th>Mapping of message bits into secret key coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 7 6 5 4 3 2</td>
<td>(k_2, k_1, k_0)</td>
<td>-2 -1 0 1 2</td>
</tr>
<tr>
<td>1 1 2 9 6 1</td>
<td>(1,153,8)</td>
<td>0 1 1 1 0</td>
</tr>
<tr>
<td>2 10 8 1 7 5 2</td>
<td>(0,77,5)</td>
<td>1 1 1 0 0</td>
</tr>
<tr>
<td>3 5 1 6 5 2 1</td>
<td>(2,335,15)</td>
<td>1 1 0 1 1</td>
</tr>
<tr>
<td>4 7 11 2 2 6</td>
<td>(3,432,3)</td>
<td>0 1 0 0 1</td>
</tr>
<tr>
<td>5 2 3 3 4 3 4</td>
<td>(3,606,3)</td>
<td>1 0 0 1 0</td>
</tr>
<tr>
<td>6 8 6 10</td>
<td>(1,864,8)</td>
<td>0 1 1 1 0</td>
</tr>
<tr>
<td>7 3 4 5 3</td>
<td>(0,915,5)</td>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>8 4 5</td>
<td>(0,898,5)</td>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>9 11 9 4 1 4 3</td>
<td>(3,432,8)</td>
<td>1 0 1 0 1</td>
</tr>
<tr>
<td>10 12</td>
<td>(0,105,5)</td>
<td>1 1 1 0 0</td>
</tr>
<tr>
<td>11 9 7 8</td>
<td>(3,632,3)</td>
<td>1 0 0 1 0</td>
</tr>
<tr>
<td>12</td>
<td>(3,386,3)</td>
<td>0 1 0 0 1</td>
</tr>
<tr>
<td>13 13</td>
<td>(3,606,8)</td>
<td>1 0 1 0 1</td>
</tr>
<tr>
<td>14 6 10 7</td>
<td>(2,321,15)</td>
<td>1 1 0 1 1</td>
</tr>
</tbody>
</table>

dataword is recovered, an error is also detected. This redundancy provides some extra error detection capability to the codes.

Ciphertexts where each decrypted message bit encodes multiple secret key coefficients, such as s_r[0] for r ∈ {0, 1, 2}, were explored. The codes using such ciphertexts would be larger and therefore more efficient, i.e., they have a smaller redundancy to error correction ratio. However, the errors that occur would no longer be independent among coefficients encoded in the same message bit. The complexity of the attack would increase but the possible reduction in traces was expected to be small. The method of encoding multiple coefficients in a single message bit was therefore excluded in this work but may be of value to some other scenario or attack.
Table 5.2: CCT construction table for Saber. [8]

<table>
<thead>
<tr>
<th>Order of codeword bits for a code with distance $d$</th>
<th>CCT constants</th>
<th>Mapping of message bits into secret key coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 5 4 3</td>
<td>$(k_1, k_0)$</td>
<td>$-4$ $-3$ $-2$ $-1$ 0 1 2 3 4</td>
</tr>
<tr>
<td>1 1 7 6</td>
<td>(240,10)</td>
<td>1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>2 5</td>
<td>(377,10)</td>
<td>0 0 1 0 1 1 0 1 0</td>
</tr>
<tr>
<td>3 7 1 4</td>
<td>(613,4)</td>
<td>1 0 1 0 1 1 0 1 0</td>
</tr>
<tr>
<td>4 2 3 1</td>
<td>(373,15)</td>
<td>1 0 1 1 0 1 0 1 0</td>
</tr>
<tr>
<td>5 10 6 3</td>
<td>(913,15)</td>
<td>1 1 1 0 0 0 0 1 1</td>
</tr>
<tr>
<td>6 11 8 7</td>
<td>(12,3)</td>
<td>1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>7 3 4 2</td>
<td>(793,10)</td>
<td>1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>8</td>
<td>(755,4)</td>
<td>0 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>9 9</td>
<td>(917,10)</td>
<td>0 1 1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>10 6</td>
<td>(806,10)</td>
<td>0 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>11 8 5</td>
<td>(456,15)</td>
<td>1 1 0 1 0 1 0 1 0</td>
</tr>
<tr>
<td>12 4 2 5</td>
<td>(68,4)</td>
<td>1 1 1 1 1 0 0 0 0</td>
</tr>
</tbody>
</table>
36 | Attack stage
Chapter 6

Results and Analysis

In this chapter, the results of the experiments are presented. The validity of the results and the results themselves are then analyzed.

6.1 Experimental results

For each of Kyber and Saber, 10 sets of data were collected for all ciphertexts, where the secret key was unique and randomly generated. Attacks using ECCs of code distance 2, 4, 6, and 8 were performed on Kyber and of 4 and 6 on Saber. For each of the ECCs, attacks using up to 30 repeated measurements for each ciphertext were used for Kyber and up to 3 for Saber.

Tables 6.1 and 6.2 show the results for Kyber and Saber, respectively, where the incorrect key coefficients are averaged over the 10 experiments and the time consumption is that of the worst case. For each combination of code distance $d$ and a number of repetitions $N$, the number of detected errors, undetected errors, capture time for the training traces on the profiling device $D_P$, capture time for the attack traces on the target device $D_A$ and estimated maximum enumeration time are presented.

Detected errors are the number of coefficients that the attacker could not recover, but knows they could not be recovered. If the number is small, they can be efficiently enumerated with a maximum of $5^e$ or $9^e$ decryptions, for Kyber and Saber respectively, for $e$ detected errors. The time to perform these decryptions have been estimated for a single-threaded implementation on a computer with a processor running at 2.2GHz. In the tables, “-” means that enumeration is not practically feasible.

Undetected errors mean that the coefficient is believed to have been recovered during the attack but is not the same as the coefficient in the true
Table 6.1: Success rate of Kyber secret key recovery using an ECC with code distance $d$ and $N$ repetitions for 10 attacks. [8]

<table>
<thead>
<tr>
<th>$N$</th>
<th>$d$</th>
<th># Incorrect key coeff. (mean)</th>
<th>Attack time (the worst case)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Undetected</td>
<td>Detected</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>17.3</td>
<td>0.9</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.1</td>
<td>5.4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>29.1</td>
<td>3.0</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>0</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.1</td>
<td>9.1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.4</td>
<td>27.8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>73.1</td>
<td>15.8</td>
</tr>
</tbody>
</table>

Table 6.2: Success rate of Saber secret key recovery using an ECC with code distance $d$ and $N$ repetitions for 10 attacks. [8]

<table>
<thead>
<tr>
<th>$N$</th>
<th>$d$</th>
<th># Incorrect key coeff. (mean)</th>
<th>Attack time (the worst case)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Undetected</td>
<td>Detected</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.4</td>
<td>5.7</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>0</td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.7</td>
<td>13.2</td>
</tr>
</tbody>
</table>

secret key. Undetected errors can not be enumerated and is therefore the most critical metric of the results. Of the time required for the attack steps, capturing the attack traces from the target device $D_A$ is the most critical since it is the only one that requires access time to the target. The capturing from the profiling device $D_P$ and enumeration can be performed offline.

The number of traces to succeed with the attack in all 10 experiments, for
Table 6.3: The number of traces required for successful key recovery in all 10 attacks. [8]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Code distance d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Kyber</td>
<td>Traces</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Saber</td>
<td>Traces</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 6.1: Average undetected and detected errors depending on the number of repeated measurements $N$ for each code distance on Kyber.

Each of the codes, are listed in Table 6.3. The numbers represent the total number of traces when increasing $N$ until all 10 experimental attacks result in no undetected errors and practically feasible enumeration. We define feasible enumeration as less than $2^{32}$ possibilities although most cases are significantly lower. The table shows that the ECC with code distance 6 is optimal for both algorithms. For Saber, as can be observed in Figure 6.2, complete elimination of undetected errors was achieved with a single repetition using code distance 6. This is the reason why code distance 8 was not evaluated. Since no fewer than one repetition was possible and the number of ciphertexts would have increased, the total number of traces would have been greater.

The Figures 6.1 and 6.2 show the decrease in undetected and detected errors as the repetitions of measurements are increased. While it is clear that the errors decrease with greater code distance, the optimal code, requiring the smallest number of traces, must additionally take the codeword length $l$ into account as it represents the number of CCTs needed.
Validity of results

The ChipWhisperer platform is designed to lower the bar for performing side-channel research and evaluations. Results on the framework are not directly comparable to real-world scenarios as they have been specifically designed to minimize noise. It does however facilitate comparison between work performed by different researchers as their hardware and the experimental conditions are similar. The results of the experiments in this thesis are therefore to be seen as results in “best case” conditions. They show the possibility of successful attacks if the attacker has sufficiently low noise and is able to gather enough attack traces.

From an academic perspective, the results are therefore valid and valuable. They should however be framed along with the context and conditions in which they were acquired.

Analysis

For Saber, the attacks can be performed with only 4,608 traces. Compared to the 61,680 traces needed using the bit-flipping technique in the attack of [4], the improvement is 13-fold. This significant reduction is achieved even though, in contrast to the work in [4]: (1) no traces from the target device were used during profiling; (2) the attack made use of only one message leakage point instead of two; (3) the best performing models were not manually picked. (1) is particularly important since the attacker can perform the entire profiling offline and further reduce the access needed to the target device. It indicates
that the attack is more robust against device diversity.

The reasons for the improved results are believed to be manifold:

- **Two-bit recovery:** Two message bits are recovered at a time, parallelizing the recovery by a factor of 2. This reduces the number of ciphertexts per message from 257 to 128.

- **Powerful ECCs:** ECCs of greater code distance are used. The increased error correction and detection capabilities that come with increasing the code distance help reduce the repetitions $N$ at a faster rate than the codeword length $l$ increases until the optimal code is found.

- **Higher recovery probability:** The probability of correctly recovering the index and its corresponding message bit is greater than recovering all of the message bits, which was required for [4]. The improved probability of recovery further reduces $N$ and likely improves robustness.

- **Higher sampling rate:** For Saber, the sampling rate was increased by a factor of 3. Increasing the level of information in the traces. This likely helped the models learn and predict with higher accuracy.

An interesting observation is that the attack performs more efficiently on the Saber implementation than on Kyber. The traces for Kyber were captured at a third of the sampling rate which could partly explain the higher number of repetitions needed. However, the most likely reason is that the leakage of the vulnerability is, in contrast to the one used in Saber, a non-uniform accumulative leakage. As has previously been shown, accumulative leakage is harder to exploit unless the already accumulated data can be compensated for. This is of course not possible in the current setting where the accumulation order is shuffled and only the bits with indexes 0 and 255 can be recovered.

As has been previously highlighted, the impact of having undetected errors is substantially worse than having detected errors. If they are detected they can, granted they are sufficiently few, easily be enumerated. If the detected errors are too many to enumerate, as long as there are no undetected errors, lattice-based methods could be applicable to recover the remaining coefficients more efficiently. By only considering the remaining coefficients, the problem can instead be seen as an LWE/LWR problem of lower dimension. It is further possible that more efficient cryptological methods exist even in the case where few undetected errors occur.
Chapter 7

Conclusions and Future work

This chapter presents the conclusions drawn from the presented work and gives suggestions for future work and research.

7.1 Conclusions

In this report, power SCA on the Fisher-Yates shuffling algorithm is presented. The results show that the Hamming weights of the indexes can be recovered; directly leading to the recovery of the first and last index, 0 and 255, as they are the only values within their weight. The results of this analysis were then used in demonstrating successful attacks on software implementations of the M-LWE and M-LWR schemes Kyber and Saber, extracting their secret keys. The attacks utilized the recovered indexes and is the first to exploit the cyclic rotation ciphertext malleability in a secret key recovery. The attack on Saber improved 13-fold upon the previous attack on the same implementation, reducing the traces required for the attack stage from 61,680 to 4,608. With this, both of the goals defined in Section 1.4 have been accomplished.

The results indicate that masking and shuffling are not necessarily enough to secure an implementation. At least not unless the implementation of the shuffling permutation generation algorithm itself is protected.

The idea of recovering only the values with all-0 and all-1 binary representations and rotating the message may find further use in other contexts. By exploiting e.g., ciphertext malleabilities, as was used in this work, to rotate the message, the ability to recover only certain values can be sufficient to attack an implementation.

Our code is available at https://github.com/lbacklund/SCA-Masked-Shuffled-Saber-Kyber.
7.2 Future work

In this work, SCA on the Fisher-Yates shuffling algorithm was performed, demonstrating the standard, modern implementation to be vulnerable. Future work could involve the protection of the Fisher-Yates permutation generation itself through means such as masking. Other methods for generating the randomized indexes could be investigated as well. It is possible that hardware implementations of permutation generators or shuffling algorithms are desirable as a complement to the common random number generators. Such an architectural component could potentially reduce the power consumption and therefore the leakage of the permutation generation.

The results of the SCA were utilized in an attack on software implementations of Kyber and Saber. It would be interesting to investigate its portability to hardware implementations. Additionally, exploring its applicability to protected implementations of other algorithms could be valuable.

As the attack presented is made possible by leakage from the chosen ciphertexts, countermeasures that prevent CCAs without relying on the FO transform should be developed. The fact that the IND-CCA2 security of the respective KEMs can be bypassed through side-channels implies that measures need to be taken before the decryption begins executing. As suggested by [26], discarding ciphertexts that appear specifically structured or have low entropy could potentially prevent or complicate CCAs that make use of sparse ciphertexts, as in the attack presented in this work.

Another perspective that may be worthwhile pursuing is developing protocols where the KEMs or the PKE algorithms are used, without exclusively relying on the confidentiality of a secret key on a device in an uncontrolled environment. This work assumes a non-authenticated key exchange where the device holding the secret key can be physically accessed. Invalidating these assumptions could potentially mitigate the threat of the attack presented in this work.
References


rc.nist.gov/projects/postquantum-cryptography/round-3-submissions
[Page 6.]

attacks on masked and shuffled implementations of CRYSTALS-Kyber
[Online]. Available: https://eprint.iacr.org/2022/1692 [Pages ix, x, xi,
6, 7, 16, 17, 23, 24, 25, 26, 27, 34, 35, 38, and 39.]

message leakage in (few) NIST PQC candidates for practical message
[Pages 7, 11, 12, 29, 30, and 31.]

RSA, DSS, and other systems,” in Proceedings of the 16th Annual
International Cryptology Conference on Advances in Cryptology, ser.
3540615121 p. 104–113. [Pages 8 and 9.]

397. [Pages 8 and 9.]

on AES using deep learning,” in Proceedings of the 4th ACM Workshop
on Attacks and Solutions in Hardware Security, ser. ASHES’20. New
York, NY, USA: Association for Computing Machinery, 2020. doi:
Available: https://doi.org/10.1145/3411504.3421214 [Page 8.]

leakage model,” in Cryptographic Hardware and Embedded Systems -
[Page 9.]


